Altera’s Avalon Communication Fabric

Stephen A. Edwards

Columbia University

Spring 2024
Altera’s Avalon Bus

Something like “PCI on a chip”

Described in Altera’s *Avalon Memory-Mapped Interface Specification* document

Protocol defined between peripherals and the “bus” (actually a fairly complicated circuit)
Controller and Responder

Most bus protocols draw a distinction between

**Controllers:** Can initiate a transaction, specify an address, etc. E.g., the Nios II processor

**Responders:** Respond to requests from controllers, can generate return data. E.g., a video controller

Most peripherals are responders

Controllers speak a more complex protocol

Bus arbiter decides which controller gains control
The Simplest Responder Peripheral

Avalon-MM Peripheral

Basically, “latch when I’m selected and written to.”
For a 16-bit connection that spans 32 halfwords,

Responder Signals

<table>
<thead>
<tr>
<th>Responder</th>
<th>Avalon</th>
</tr>
</thead>
<tbody>
<tr>
<td>← clk</td>
<td>⇒</td>
</tr>
<tr>
<td>← reset</td>
<td>→</td>
</tr>
<tr>
<td>← chipselect</td>
<td></td>
</tr>
<tr>
<td>← address[4:0]</td>
<td></td>
</tr>
<tr>
<td>← read</td>
<td></td>
</tr>
<tr>
<td>← write</td>
<td></td>
</tr>
<tr>
<td>← byteenable[1:0]</td>
<td></td>
</tr>
<tr>
<td>← writedata[15:0]</td>
<td></td>
</tr>
<tr>
<td>readdata[15:0]</td>
<td></td>
</tr>
<tr>
<td>irq</td>
<td></td>
</tr>
</tbody>
</table>
Avalon Responder Signals

- **clk**: Main clock
- **reset**: Reset signal to peripheral
- **chipselect**: Asserted when bus accesses peripheral
- **address[..]**: Word address (data-width specific)
- **read**: Asserted during peripheral → bus transfer
- **write**: Asserted during bus → peripheral transfer
- **writedata[..]**: Data from bus to peripheral
- **byteenable[..]**: Indicates active bytes in a transfer
- **readdata[..]**: Data from peripheral to bus
- **irq**: peripheral → processor interrupt request

All are optional, as are many others for, e.g., flow-control and burst transfers.
In SystemVerilog

```verilog
class myresponder(input logic clk, input logic reset, input logic [7:0] writedata, input logic write, input logic chipselect, input logic [2:0] address);
```
Bus cycle starts on rising clock edge
Data latched at next rising edge
Such a peripheral must be purely combinational
Responder Read Transfer w/ 1 Wait State

Bus cycle starts on rising clock edge
Data latched two cycles later
Approach used for synchronous peripherals
Basic Async. Responder Write Transfer

Bus cycle starts on rising clock edge
Data available by next rising edge
 Peripheral may be synchronous, but must be fast
Bus cycle starts on rising clock edge
Peripheral latches data two cycles later
For slower peripherals
The Vga_ball Peripheral

module vga_ball(input logic clk,
                 input logic reset,
                 input logic [7:0] writedata,
                 input logic write,
                 input chipselect,
                 input logic [2:0] address,
                 output logic [7:0] VGA_R, VGA_G, VGA_B,
                 output logic VGA_CLK, VGA_HS, VGA_VS,
                 output logic VGA_BLANK_n,
                 output logic VGA_SYNC_n);

logic [10:0] hcount;
logic [9:0] vcount;
logic [7:0] background_r, background_g, background_b;

vga_counters counters(.clk50(clk), .*);
```verilog
always_ff @(posedge clk)
  if (reset) begin
    background_r <= 8'h0;
    background_g <= 8'h0;
    background_b <= 8'h80;
  end else if (chipselect && write)
    case (address)
      3'h0 : background_r <= writedata;
      3'h1 : background_g <= writedata;
      3'h2 : background_b <= writedata;
    endcase

always_comb begin
  {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
  if (VGA_BLANK_n )
    if (hcount[10:6] == 5'd3 &&
        vcount[9:5] == 5'd3)
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
  else
    {VGA_R, VGA_G, VGA_B} =
      {background_r, background_g, background_b};
end
```